Universidad Nacional de San Agustín de Arequipa

Escuela Profesional:
Ciencias de la Computación
Tema:
Octree y Cuantización de color
Alumnos:
Renzo Vicente Castro
Luis Villanueva Flores
Curso:
Estructuras de Datos Avanzadas
Docente:
Vicente Machaca Arceda

2019

Octree y Cuantización del color

Renzo Vicente Castro - Luis Villanueva Flores

Estructuras de Datos Avanzadas

1. Introducción:

En el siguiente documento se tratará una de las principales aplicaciones de la estructura de datos Octree, la cuantización del color.

2. ¿Qué es un Octree?

Un octree o árbol octal es una estructura en "árbol"de datos en la cual cada nodo interno tiene exactamente 8 "hijos". Las estructuras octree se usan mayormente para particionar un espacio tridimensional, dividiéndolo recursivamente en ocho octantes. Las estructuras octree son las análogas tridimensionales de los quadtree bidimensionales. El nombre está formado a partir de oct (octante) + tree (árbol), y normalmente se escribe como .ºctree.en vez de .ºcttree".[1]

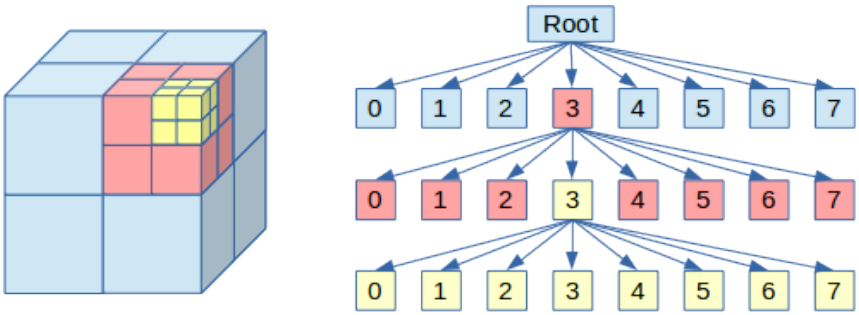

Figura 1: Representación de un Octree.

3. Estructuras octree para la representación espacial :

En una estructura octree, cada nodo subdivide el espacio que representa en ocho octantes. En una región punto (PR) octree, el nodo almacena un punto

tridimensional explícito, el cual es el çentro"de la subdivisión para ese nodo; el punto que define una de las esquinas para cada uno de los ocho hijos. En una octree MX, el punto de subdivisión es implícitamente el centro del espacio que el nodo representa. El nodo raíz de una PR octree puede representar un espacio infinito; el nodo raíz de una octree MX debe representar un espacio con límite finito para que los centros implícitos estén bien definidos. Las estructuras octree nunca se consideran árbol kd, ya que los árboles kd dividen en una dimensión mientras que las estructuras octree dividen alrededor de un punto. Los árboles kd además son siempre binarios, lo cual no se cumple para las estructuras octree.[1]

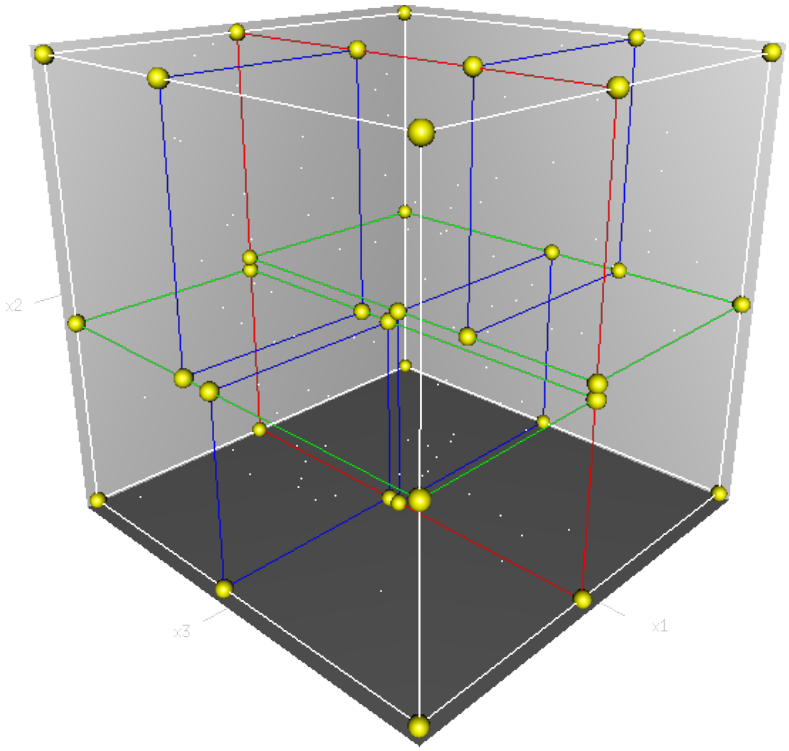

Figura 2: Un árbol kd no se considera una estructura Octree.

4. Aplicaciones:

- Detección de colisiones eficiente en tres dimensiones.[1]
- Cuantificación de color.[1]
- Teorema de Bayes.[1]
- Determinación de cara oculta.[1]

- Método multipolo rápido.[1]
- Métodos no estructurados.[1]
- Procesamiento de imágenes.[1]

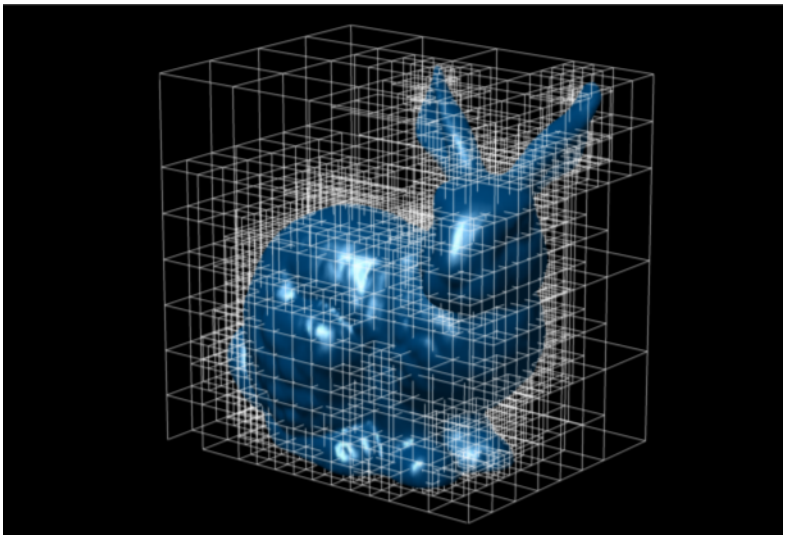

Figura 3: El procesamiento de imágenes es una de las principales aplicaciones del Octree.

5. Cuantización de color:

La cuantización del color reduce el número de colores usados en una imagen; esto es importante para visualizar imágenes en dispositivos que soportan un número limitado de colores y para eficiencia de compresión de ciertos tipos de imágenes. La mayoría de los editores de imagen y muchos sistemas operativos soportan de forma nativa la cuantización del color. Entre los algoritmos de cuantización de color modernos más populares se encuentran el algoritmo del color más cercano (para paletas fijas), el algoritmo Median cut, y un algoritmo basado en octrees.[2]

5.1. Cuantificación de color con Octrees:

El octree de color de cuantización algoritmo, inventado por Gervautz y Purgathofer en 1988, codifica los datos de imagen de color como un octree hasta nueve niveles de profundidad. Octrees se utilizan porque y hay tres componentes de color en el RGB sistema. El índice de nodo a ramificarse a partir de en el nivel superior se determina por una fórmula que utiliza los bits más significativos

de los componentes de color rojo, verde, y azul, por ejemplo 2g 4r + + b. El siguiente nivel inferior utiliza el siguiente significado bits, y así sucesivamente. Bits significativos menos veces no se tienen que reducir el tamaño del árbol. 2 ^{3} = 8.[3]
El algoritmo es altamente eficiente de la memoria, porque el tamaño del árbol puede ser limitada. El nivel inferior de la octree consta de nodos de hoja que se acumulan los datos de color que no están representados en el árbol; estos nodos inicialmente contienen los bits individuales. Si se introducen mucho más que el número deseado de colores de la paleta en el octree, su tamaño puede reducirse continuamente por la búsqueda de un nodo de nivel inferior y promediando los datos de bits de hasta en un nodo hoja, poda parte del árbol. Una vez que el muestreo se completa, la exploración de todas las rutas en el árbol hacia abajo para los nodos hoja, tomando nota de los bits en el camino, producirá aproximadamente el número requerido de colores.[3]

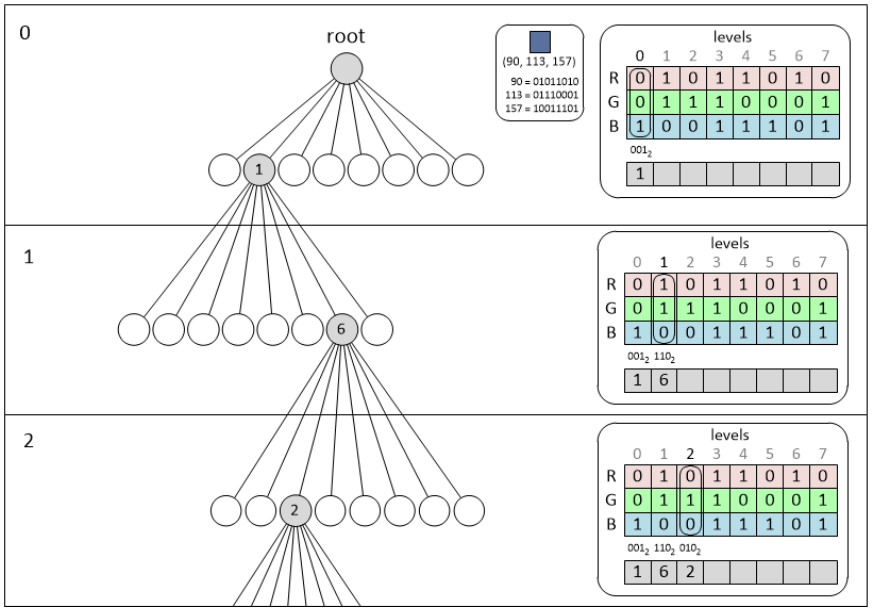

Figura 4: Cuantificación de colores con Octrees.

Figura 5: Imagen sin reducciones - imagen con reducciones.

6. Implementación:

6.1. Clase color e inclusiones para el proyecto

Listing 1: Clase color

```
#include "pch.h"
#include <opencv2/opencv.hpp>
#include <vector>

using namespace cv;
using namespace std;
class Color
{
public:
        int red, green, blue, alpha;
        Color(int red = 0, int green = 0, int blue = 0, int
            alpha = 1)
        {
                this->red = red;
                this->green = green;
                this->blue = blue;
                this->alpha = alpha;
        }
};
```

En esta parte del código lo que hacemos es incluir las librerías que usaremos en nuestro código y también definimos la clase color, que tiene como valores al rojo, azul , verde y alpha que es la opacidad, y definimos su constructor por parámentro.

6.2. Clase Octree Node

Listing 2: Clase OctreeNode

```
1  class OctreeQuantizer;
2  class OctreeNode
3  {
4  public:
5          int pixel_count, palette_index;
6          Color color;
7          vector<OctreeNode*>children;
8          OctreeNode(int level, OctreeQuantizer *parent);
9          //friend class OctreeQuantizer;
10         bool is_leaf();
11         vector<OctreeNode*>get_leaf_nodes();
12         int get_nodes_pixel_count();
13         int get_palette_index(Color color, int level);
14         int remove_leaves();
15         int get_color_index_for_level(Color color, int level);
16         Color get_color();
17         void add_color(Color color, int level, OctreeQuantizer *
               parent);
18
19 };
```

En esta parte definimos la clase OctreeQuantizer debido a que nuestra clase OctreeNode también la necesita, y en ella declaramos lo que usaremos como el pixelcount y el paletteindex que nos servirá para contar nuestro pixeles y saber el índice de algun pixel que querramos, también tenemos un elemento color que nos dirá que color es el que guarda, así como las funciones que usaremos, como el removeleaves que nos removerá las hojas , también para añadir el color, entre otras que se irán explicando mas adelante.

6.2.1. Funciones Clase Octree Node

Listing 3: Constructor OctreeNode

```
1  OctreeNode::OctreeNode(int level, OctreeQuantizer*parent)
2  {
3          this->pixel_count = 0;
4          this->palette_index = 0;
5          this->color.red = 0;
6          this->color.blue = 0;
7          this->color.green = 0;
8          this->children.resize(8);
9          if (level < parent->MAX_DEPTH-1)
10         {
11                 parent->add_level_node(level, this);
12         }
13
14 }
```

En estas parte de código definimos todas las funciones que usará nuestra clase Octree Node , así como su contructor que lo inicilizamos con cero al pixel-count al paletteindex así como a sus colores propios, y a sus hijos les damos un tamaño de 8.

Listing 4: Funciones OctreeNode

```
1  bool OctreeNode::is_leaf()
2  {
3          return this->pixel_count > 0;
4  }
5  vector<OctreeNode*>OctreeNode::get_leaf_nodes()
6  {
7
8          vector<OctreeNode*>leaf_nodes;
9          for (int i = 0; i < 8; i++)
10         {
11                 OctreeNode *Node = this->children[i];
12                 if (Node != nullptr)
13                 {
14                         if (Node->is_leaf())
15                         {
16                                 leaf_nodes.push_back(Node);
17                         }
18                         else
19                         {
20                                 vector<OctreeNode*> vector2 =
                                       Node->get_leaf_nodes();
21                                 leaf_nodes.insert(leaf_nodes.end
                                       (), vector2.begin(), vector2
                                       .end());
22                         }
23                 }
24
25         }
26         return leaf_nodes;
27 }
28 int OctreeNode::get_nodes_pixel_count()
29 {
30         int sum_count;
31
32         sum_count = this->pixel_count;
33         for (int i = 0; i < 8; i++)
34         {
35                 OctreeNode *Node = this->children[i];
36                 if (Node!=nullptr)
37                 {
38                         sum_count += Node->pixel_count;
39                 }
40         }
41         return sum_count;
42 }
```

Aquí definimos una de sus funciones así como verificar si es hoja, también obtener en un vector todas las hojas y retornar todas aquellas que lo sean, y también

la función que nos obtiene la suma total de nuestro pixelcount que básicamente recorrera todos lo hijos y sumará su pixelcount de cada uno.

Listing 5: Funciones OctreeNode

```
int OctreeNode::get_palette_index(Color color, int level)
{
        if (this->is_leaf())
        {
                return this->palette_index;
        }
        int index = this->get_color_index_for_level(color, level
            );
        if (this->children[index])
        {
                return this->children[index]->get_palette_index(
                    color, level + 1);
        }
        else
        {
                for (int i = 0; i < 8; i++)
                {
                        if (this->children[i] != nullptr)
                        {
                                return this->children[i]->
                                    get_palette_index(color,
                                    level + 1);
                        }
                }
        }
}
int OctreeNode::remove_leaves()
{
        int result = 0;

        for (int i = 0; i < 8; i++)
        {
                OctreeNode *Node;
                Node = this->children[i];
                if (Node!=nullptr)
                {
                        this->color.red += Node->color.red;
                        this->color.green += Node->color.green;
                        this->color.blue += Node->color.blue;
                        this->pixel_count += Node->pixel_count;
                        result += 1;


                }
        }
        return result - 1;

}
```

En estas funciones obtenemos en nuestra paleta el index osea la posición que

tendra en nuestra imagen algún color, y en la función posterior lo que hacemos es remover las hojas pero vamos sumando los colores para acumularlos.

Listing 6: Funciones OctreeNode

```
int OctreeNode::get_color_index_for_level(Color color, int level
    )
{
        int index = 0;
        int mask = 0x80 >> level;
        if (color.red & mask)
        {
                index |= 4;
        }
        if (color.green & mask)
        {
                index |= 2;
        }
        if (color.blue & mask)
        {
                index |= 1;
        }
        return index;
}
Color OctreeNode::get_color()
{
        return Color(
                this->color.red / this->pixel_count,
                this->color.green / this->pixel_count,
                this->color.blue / this->pixel_count);
}
void OctreeNode::add_color(Color color, int level,
    OctreeQuantizer *parent)
{
        if (level >= parent->MAX_DEPTH)
        {
                this->color.red += color.red;
                this->color.green += color.green;
                this->color.blue += color.blue;
                this->pixel_count += 1;
                return;
        }
        int index = this->get_color_index_for_level(color, level
            );
        if (children[index] == nullptr)
        {
                this->children[index] = new OctreeNode(level,
                    parent);
        }
        this->children[index]->add_color(color, level + 1,
            parent);
}
```

En esta primera función obtenemos el número de un color lo convertimos a binario y retornamos el número que deberá ir en la posición de nuestro Octree.

Después definimos nuestra función de obtener el color, que lo que hace es a cada color lo divide entre el contador de pixel para sacar un promedio.

Después hacemos la función de añadir el color que lo que hace es recorrer nuestro octree apartir del padre y hasta un nivel y va sumando a la hoja todo aquel color que sea igual para ir acumulandolo.

6.3. Clase OctreeQuantizer

Listing 7: Clase OctreeQuantizer

```
1  class OctreeQuantizer
2  {
3  public:
4
5          int MAX_DEPTH = 8;
6          OctreeNode *root;
7          //friend class  OctreeNode;
8          vector<vector<OctreeNode*>>levels;
9          OctreeQuantizer()
10         {
11                 this->levels.assign(this->MAX_DEPTH, {});
12                 this->root = new OctreeNode(0, this);
13         }
14         vector<OctreeNode*>get_leaves()
15         {
16                 return this->root->get_leaf_nodes();
17         }
18         void add_level_node(int level, OctreeNode *node)
19         {
20                 this->levels[level].push_back(node);
21         }
22
23         void add_color(Color color)
24         {
25                 this->root->add_color(color,0,this);
26         }
27         vector<Color> make_palette(int color_count) {
28                 vector<Color> palette;
29                 int palette_index = 0;
30                 int leaf_count = this->get_leaves().size();
31                 for (int level = this->MAX_DEPTH - 1; level >
                       -1; level -= 1) {
32                     if (this->levels[level].size() > 0) {
33                             for (auto node : this->levels[
                                    level]) {
34                                 leaf_count -= node->
                                        remove_leaves();
35                                 if (leaf_count <=
                                        color_count)
36                                 {
37                                         break;
38                                 }
39                             }
```

```
40                                  if (leaf_count <= color_count)
41                                          break;
42                                  this->levels[level].clear();
43                          }
44                  }
45                  for (auto node : this->get_leaves()) {
46                          if (palette_index >= color_count)
47                                  break;
48                          if (node->is_leaf())
49                                  palette.push_back(node->
                                        get_color());
50                          node->palette_index = palette_index;
51                          palette_index += 1;
52                  }
53                  return palette;
54          }
55          int get_palette_index(Color color)
56          {
57                  return this->root->get_palette_index(color, 0);
58          }
59  };
```

En esta clase su función principal sería a partir de la imagen analizada y con nuestro Octree poder construir una paleta de la dimensión que le pasemos a partir de la imagen original, esta clase utiliza a nuestra clase anterior, ya que usa funciones como saber si son hojas, añade un color, añade a un nivel un nodo. Crea una paleta a partir de cuantos colores querramos, y también obtiene el índice de un color en nuestra paleta, para eso usa la funciópn de nuestra clase anterior.

7. Resultados

En esta parte mostraremos los resultados obtenidos a partir de nuestra imagen original.

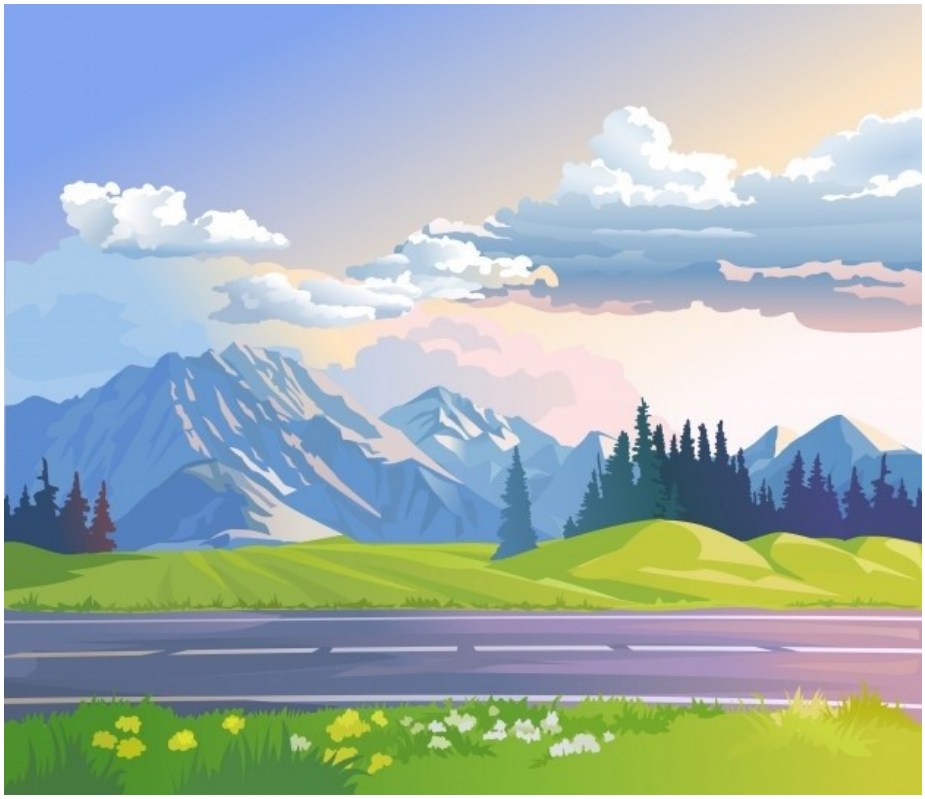

Figura 6: Imagen original sin reducciones.

Ahora aplicaremos una reducción utilizando solamente 256 colores.

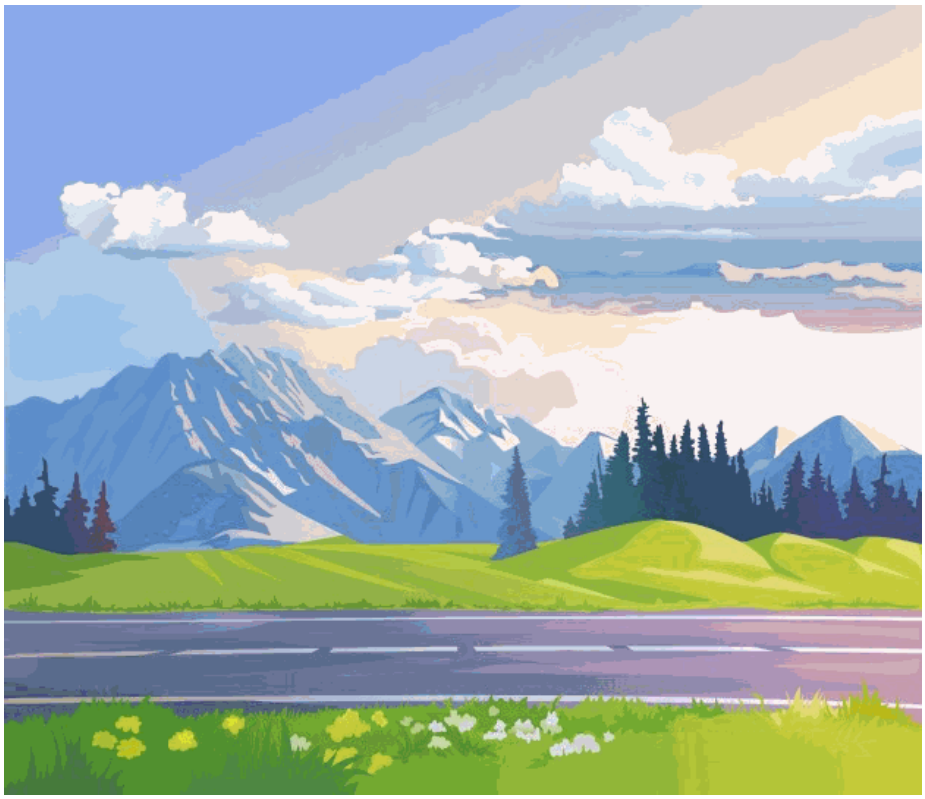

Figura 7: Imagen reducida a 256 colores .

Ahora aplicaremos una reducción utilizando solamente 64 colores.

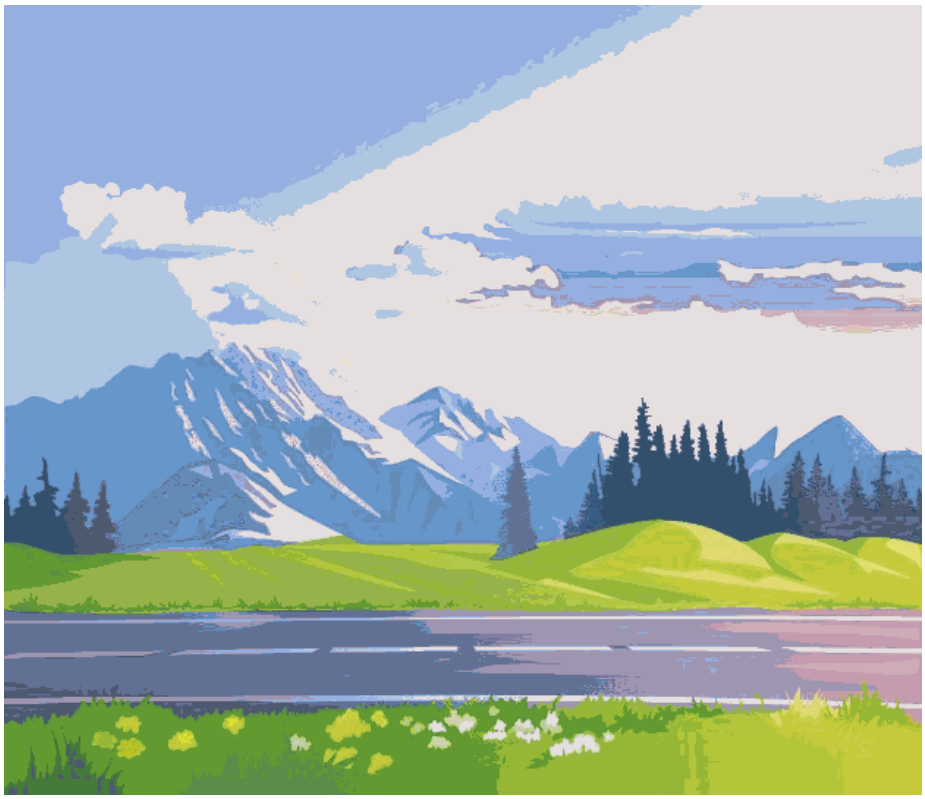

Figura 8: Imagen reducida a 64 colores .

Ahora aplicaremos una reducción utilizando solamente 4 colores.

Figura 9: Imagen reducida a 4 colores .

8. Conclusiones:

- Las reducciones del algoritmo de cuantificación de color, permiten reducir la resolución de una imagen, hace que esta pese menos también.
- Uno de los principales objetivos de la cuantización de colores es convertir una imagen en otra más similarmente visible a esta primera.
- El objetivo de la implementación es adaptar la estructura de datos Octree para que realice la aplicación de cuantizar colores.
- La librería OpenCV toma la imagen como una matriz de pixeles facilitando el manejo de las operaciones sobre esta primera.

9. Referencias:

1. Eberhardt, H., Klumpp, V., & Hanebeck, U. (2010). Density trees for efficient nonlinear state estimation. Lecture, Edinburgh, United Kingdom.

2. Baolong, G., & Xiang, F. (2006). A modified Octree color quantization algorithm. Lecture, Beijing, China.

3. Lal, P., & Jacob, K. (2001). Generación de Octrees a partir de la trama de exploración con pérdida de información reducida. Lecture, Marbella, España.